Optimization of the Controlled Evaluation of Closed Relational Queries

Joachim Biskup, Jan-Hendrik Lochner, and Sebastian Sonntag

Fakultät für Informatik, Technische Universität Dortmund, D-44221 Dortmund, Germany
{biskup,lochner,sonntag}@ls6.cs.tu-dortmund.de

Abstract. For relational databases, controlled query evaluation is an effective inference control mechanism preserving confidentiality regarding a previously declared confidentiality policy. Implementations of controlled query evaluation usually lack efficiency due to costly theorem prover calls. Suitably constrained controlled query evaluation can be implemented efficiently, but is not flexible enough from the perspective of database users and security administrators. In this paper, we propose an optimized framework for controlled query evaluation in relational databases, being efficiently implementable on the one hand and relaxing the constraints of previous approaches on the other hand.

1 Introduction

Protection of sensible information is an important issue in modern database applications. The information to be protected has to be suitably declared by the "owner" of the information or a security administrator. In this context, it is important to differentiate between *data*, which is always explicitly represented in a database instance, and *information*, which can also be obtained by applying semantics to the data. E. g., the information that Smith has an account balance of $ 15,000 can be an explicit part of the instance of a bank database, or it can be inferred, e. g., by combining the facts "Smith has the account 12345" and "The account 12345 has a balance of $ 15,000". Consequently, it may not be sufficient to protect only data but possibly also unwanted information flows have to be avoided. Thus, mechanisms only regulating the access to data may not be adequate to enforce desired protection goals.

Among other approaches, controlled query evaluation (CQE) [4] is an effective method for protecting sensible information as declared by a confidentiality policy (hereafter called "policy" for short). This method checks whether the true answer to a query together with the a priori knowledge of the user enables the user to infer any information being protected by the policy and, if necessary, modifies the answer to the query, either by lying (i. e., returning the negated answer), or by refusal (i. e., returning no answer), or by a combination of both.

CQE is a highly flexible approach that guarantees preservation of confidentiality for logic-oriented information systems. Considering relational databases, CQE is also applicable in theory, but the underlying first-order logic of relational databases is undecidable in general. For employing CQE in practical applications it is therefore necessary to restrict the first-order logic used for expressing database queries and policies to a

D. Gritzalis and J. Lopez (Eds.): SEC 2009, IFIP AICT 297, pp. 214–225, 2009.
© IFIP International Federation for Information Processing 2009

decidable fragment. Nevertheless, real database systems employing CQE would lack efficiency, because they had to rely on theorem prover calls which are known to be costly in general. These theorem prover calls result from the need of computing the inferences a database user can draw by means of his a priori knowledge about the database system and the answers to his queries. Avoiding theorem prover calls at all requires to substantially restrict the expressiveness of the query language and the policy language.

In this paper, we propose a framework that principally accepts every first-order logic sentence as a query (as long as decidability is guaranteed) but (as far as possible) eliminates costly theorem prover calls. More specifically, in Sect. 2, we briefly address approaches for the inference problem in relational databases in general and CQE in particular; in Sect. 3, we identify situations that allow for static inference control without theorem prover calls and propose flexible policy and query languages; in Sect. 4, we present SQL implementations of our static inference control; in Sect. 5, we develop an approach for an optimization framework based on the results of Sect. 3; in Sect. 6, we conclude and point out directions for future research.

2 Inference Control in Relational Databases

Security in relational databases in general and confidentiality in particular has been investigated from various perspectives. Early approaches, e. g. [16,18,22], focus on access control, which operates on the actual data and attaches access or classification information directly to this data.

Discretionary access control (DAC), whose general concept is described in popular textbooks on computer security, mainly suffers from the responsibility of the "data owner" or the security administrator to correctly assign access rights. Information disclosure by inferences cannot be controlled by DAC in general.

Mandatory access control (MAC) employs system-wide policies on classified data according to a security model; see, e. g., [20]. Among other approaches, multilevel secure databases, polyinstantiation, and various extensions have been proposed to enforce MAC; see, e. g., [13,17,18,19]. MAC is principally able to prevent unwanted information flows caused by sequences of read and write operations. Several authors propose entire frameworks, design processes, or comprehensive requirements analyses for secure database systems, e. g., [2,12]. A comprehensive overview of the inference problem in databases, the area of data mining, and Web-based applications can be found in the work of Farkas/Jajodia [15]. Further work on prevention of harmful inferences in databases has been published by Brodsky et al. [11] and Dawson et al. [14].

The first ideas of protecting information in databases according to security policies by giving lied answers or by refusing to answer at all arise from the work of Bonatti/Kraus/Subrahmanian [10] and Sicherman/de Jonge/van de Riet [21], respectively. These ideas are combined by Biskup/Bonatti to CQE, elaborated at first for logical databases [4,6,7] and extended for relational databases in [5]. Biskup/Embley/Lochner [8] propose a static form of CQE.

Beginning with some formal concepts, we now roughly sketch CQE in relational databases. A *relation schema* describes the structure of a relation in a relational database and is denoted by $\langle R, \mathscr{U}, \Sigma \rangle$ where R is the *relation symbol*, $\mathscr{U}$ is a finite set of *attributes*, and Σ is a finite set of *local semantic constraints*. We assume Σ to be a minimal

cover (see [1]) of functional dependencies. An *instance* r of a relation schema is considered as the "contents" of the relation; from a (first-order-)logic-oriented perspective (see [1]), it is a finite Herbrand interpretation of the schema satisfying Σ and considering R as a predicate. With $\mu = R(c_1,\ldots,c_n)$ we denote a *tuple*; each c_i is element of an infinite set of constants *Const* and $n = |\mathscr{U}|$. Finally, $\models_{\mathrm{M}}$ denotes the satisfaction relation between an interpretation and a formula, so if μ is element of r, we write $r \models_{\mathrm{M}} \mu$. If $\mathscr{A},\mathscr{B} \subseteq \mathscr{U}$ are attribute sets, r is said to satisfy the *functional dependency (fd)* $\mathscr{A} \to \mathscr{B}$ if any two tuples of r agreeing on the $\mathscr{A}$ values also agree on the $\mathscr{B}$ values. An attribute set $\mathscr{K}$ is a *key* of *RS* if $\Sigma \models \mathscr{K} \to \mathscr{U}$ and $\mathscr{K}$ is minimal with this property. *RS* is in *object normal form (ONF)* if it has a unique key and for each fd $\mathscr{A} \to \mathscr{B}$, logically implied by Σ and with $\mathscr{B} \not\subseteq \mathscr{A}$, $\mathscr{A}$ corresponds to this key or a superset of it [3].

Database queries are expressed in a suitable fragment of the relational calculus, meaning that each query must have a prenex normal form with prefix either $\forall^*$ or $\exists^*$; so, quantifiers may not be mixed. This condition guarantees that we do not leave the Bernays-Schönfinkel class of decidable first-order formulas [5]. Moreover, we concentrate on closed queries, i. e., we may not use free variables. The *ordinary evaluation* of a query Φ in an instance r is defined by $eval^*(\Phi)(r) := \texttt{if } r \models_{\mathrm{M}} \Phi \texttt{ then } \Phi \texttt{ else } \neg\Phi$. *Controlled query evaluation (CQE)* deviates from this ordinary evaluation if any of the previously declared potential secrets is going to be disclosed by the database user. A *potential secret* Ψ is a sentence from the policy language. If $r \not\models_{\mathrm{M}} \Psi$, the user may learn that Ψ is false in r; if, however, $r \models_{\mathrm{M}} \Psi$, the user may not learn that Ψ is actually true. The (finite) set *pot_sec*, consisting of potential secrets, denotes a *confidentiality policy* being *known* to the user. The *a priori user knowledge* log_0 is assumed to comprise Σ and possibly further sentences being true in r. It is required that $log_0 \not\models \Psi$ for each $\Psi \in pot\_sec$ and $r \models_{\mathrm{M}} log_0$ for the database instance r.

CQE for known potential secrets enforced by *(improved) refusal* is defined by $cqe(Q,log_0)(r,pot\_sec) := \langle (ans_1,log_1),(ans_2,log_2),\ldots\rangle$ for a query sequence $Q = \langle \Phi_1,\Phi_2,\ldots\rangle$. It uses a *censor function* to determine the returned answer ans_i (with `mum` denoting a refusal) and the current user knowledge log_i for each query, and preserves confidentiality in the sense of the following Def. 1 (see [6,7]).

$$
\begin{aligned}
&censor(pot\_sec,log,\Phi) := \qquad (1)\\
&\quad (\text{exists } \Psi)(\Psi \in pot\_sec \text{ and } (log \cup \{\Phi\} \models \Psi \text{ or } log \cup \{\neg\Phi\} \models \Psi))\\
&ans_i := \texttt{if } log_{i-1} \models eval^*(\Phi_i)(r) \texttt{ then } eval^*(\Phi_i)(r)\\
&\quad \texttt{else if } censor(pot\_sec,log_{i-1},\Phi_i) \texttt{ then mum else } eval^*(\Phi_i)(r)\\
&log_i := \texttt{if } censor(pot\_sec,log_{i-1},\Phi_i) \texttt{ then } log_{i-1}\\
&\quad \texttt{else } log_{i-1} \cup \{eval^*(\Phi_i)(r)\}.
\end{aligned}
$$

Definition 1 (Confidentiality preservation). *A CQE is* confidentiality preserving for *pot_sec if for every finite prefix Q' of a query sequence Q the following holds: For every $\Psi \in pot\_sec$, for every instance r_1, and for every a priori knowledge log_0 there exists an instance r_2 with $r_2 \models_{\mathrm{M}} log_0$ and*

(1) $cqe(Q',log_0)(r_1,pot\_sec) = cqe(Q',log_0)(r_2,pot\_sec)$ and
(2) $eval^*(\Psi)(r_2) = \neg\Psi$.

A CQE is confidentiality preserving *if it is confidentiality preserving for all possible confidentiality policies.*

The above CQE definition is highly flexible, since it works for queries and secrets expressed in any compact logic with a suitably defined "model-of" operator. However, a general drawback of this approach is the (costly) computation of inferences each time the censor is invoked. The censor decision (1) can be reduced to a NEXPTIME complete satisfiability problem.

Biskup/Embley/Lochner [8] identify a parameter configuration allowing for simpler inference computations in form of pattern matching. Their approach roughly imposes the following restrictions to database schema, query language and policy language. The database schema has to be in ONF. The query language $\mathscr{L}_{\mathrm{q}}$ is restricted to *existential-R-sentences*, i. e., closed formulas of the positive existential calculus [1] without logical connectives. Each query has the form $\Phi \equiv (\exists X_1)\ldots(\exists X_m)R(v_1,\ldots,v_n)$ with $v_i = X_j$ or $v_i \in \mathit{Const}$ and each X_i occurring exactly once in $v_1,\ldots,v_n$. The policy language $\mathscr{L}_{\mathrm{ps}}$ is also restricted to existential-R-sentences; moreover, each potential secret Ψ must protect a *fact* of the schema, i. e., the constants in Ψ must instantiate the unique key and at most one additional attribute. E. g., a schema with $\mathscr{U} = \{A,B,C\}$ and $\Sigma = \{A \to BC\}$ has the *fact schemas* A, AB, and AC; thus, $(\exists X_B)(\exists X_C)R(c_A,X_B,X_C)$, $(\exists X_C)R(c_A,c_B,X_C)$, and $(\exists X_B)R(c_A,X_B,c_C)$ are proper potential secrets (with X_B, X_C being variables and $c_A,c_B,c_C,c_D \in \mathit{Const}$). These restrictions lead to the following *static* censor that is independent of the user log, which therefore needs not to be considered any longer.

$$\mathit{censor}_{\mathit{stat}}(\mathit{pot\_sec},\Phi) := (\text{exists } \Psi)(\Psi \in \mathit{pot\_sec} \text{ and } \Phi \models \Psi) \tag{2}$$

We denote the CQE using $\mathit{censor}_{\mathit{stat}}$ by $\mathit{cqe}_{\mathit{stat}}$. In [8] it is proved that $\mathit{cqe}_{\mathit{stat}}$ preserves confidentiality in the sense of Def. 1.

Biskup/Lochner [9] propose an algorithm with logarithmic runtime that can easily be adapted to $\mathit{cqe}_{\mathit{stat}}$. This algorithm performs a pattern matching between the query Φ and each potential secret Ψ. If and only if Φ and (at least one) Ψ agree on each constant in Ψ, $\Phi \models \Psi$ holds and `mum` is returned.

3 Optimizing Static Inference Control for Closed Queries

Unfortunately, we achieve the confidentiality preserving static inference control introduced in Sect. 2 only at the expense of the expressiveness of the underlying languages. The objective of this section is to identify relaxations of the restrictions while keeping up static inference control.

Inference control in relational databases in general and CQE in particular offer a variety of parameters. We confine ourselves to the following: Policies are supposed to consist only of potential secrets and to be known to database users. We believe that functional dependencies are the most important and prevalent type of (local) semantic constraints and therefore neglect other types of local semantic constraints, and global semantic constraints (like inclusion dependencies) as well. Thus, the relations of a database are independent of each other; so, for simplicity, we assume a database to

consist of exactly one relation schema. We consider a single database user (besides the security administrator) and concentrate on the (improved) refusal method for enforcing policies. We assume languages $\mathscr{L}_{\mathrm{q}}^{max}$ and $\mathscr{L}_{\mathrm{ps}}^{max}$ as "upper bounds" for the query language and the policy language, respectively. Each element of $\mathscr{L}_{\mathrm{q}}^{max}$ is a sentence of the form $\exists^* \varphi$ with $\exists^*$ being a sequence of existentially quantified variables and φ being a quantifier-free first-order formula, i. e., a Boolean combination of R-atoms. Each existentially quantified variable is supposed to occur only once in φ. Elements of $\mathscr{L}_{\mathrm{ps}}^{max}$ may additionally contain free variables. Again, each free variable is supposed to occur only once in a formula from $\mathscr{L}_{\mathrm{ps}}^{max}$.

For illustrating our investigations we hereafter refer to the following example.

Example 1. A (fictitious) group of banks maintains a common database for administrating information about the account holders. For each combination of bank and account number the account holder and the balance of the account are stored in the database. Let the schema of this database be given by $\langle bank\_db, \mathscr{U}, \Sigma \rangle$ with $\mathscr{U} = \{bank, acc\_no, acc\_holder, balance\}$ and $\Sigma = \{bank, acc\_no \rightarrow acc\_holder, balance\}$. Obviously, *bank_db* is in ONF with the key $\mathscr{K} = \{bank, acc\_no\}$. This yields the set of fact schemas $fs(bank\_db) = \{\{bank, acc\_no\}, \{bank, acc\_no, acc\_holder\}, \{bank, acc\_no, balance\}\}$. Consider this instance of *bank_db*:

bank_db	*bank*	*acc_no*	*acc_holder*	*balance*
	Bank of Springfield	123654	Smith	$ 15,000
	Gotham City Bank	213456	Jones	$ 2,500
	Metropolis Financial Group	321645	Parker	$ 100
	Gotham City Bank	312564	Smith	$ 2,500
	Bank of Springfield	213456	Green	$ 15,000

Suppose that the group of banks outsources the statistical evaluation of their accounts to an external service provider. In doing so, certain information should be kept secret, e.g., the association between an account number and the corresponding account holder. Thus, a policy *pot_sec* is defined and enforced by a CQE.

3.1 The Query Language

In [8], the query language $\mathscr{L}_{\mathrm{q}}$ ($\subseteq \mathscr{L}_{\mathrm{q}}^{max}$) is introduced, which is restricted to existential-R-sentences. With this language it is possible to ask for (full) tuples or for subtuples (i. e., parts of tuples) only; thus we can express queries like

$\Phi_1 \equiv bank\_db(\text{Gotham City Bank}, 213456, \text{Jones}, 2500)$ and
$\Phi_2 \equiv (\exists X_{acc})(\exists X_{bal}) bank\_db(\text{Bank of Springfield}, X_{acc}, \text{Parker}, X_{bal})$.

Adding disjunction to $\mathscr{L}_{\mathrm{q}}$ may cause problems as shown by the following example.

Example 2. Consider the following policy, meaning that the user may not learn that the Bank of Springfield maintains an account with the number *123654*:

$pot\_sec = \{(\exists X_{hold})(\exists X_{bal}) bank\_db(\text{Bank of Springfield}, 123654, X_{hold}, X_{bal})\}$.

The user with the a priori knowledge $log_0 = \emptyset$ now poses two queries using $\mathscr{L}_{\mathrm{q}}$ enhanced with disjunction:

$$\Phi_1 \equiv (\exists X_{hold})(\exists X_{bal})bank\_db(\text{Bank of Springfield}, 123654, X_{hold}, X_{bal}) \vee (\exists X_{bank})(\exists X_{acc})(\exists X_{bal})bank\_db(X_{bank}, X_{acc}, \text{Scott}, X_{bal})$$
$$\Phi_2 \equiv (\exists X_{bank})(\exists X_{acc})(\exists X_{bal})bank\_db(X_{bank}, X_{acc}, \text{Scott}, X_{bal})$$

The CQE with the censor (2) answers Φ_1 as well as Φ_2 correctly because neither of them directly implies the potential secret. However, since Φ_1 is true and Φ_2 is false in *bank_db*, the combination of both answers implies the secret.

The sketched problem is inherent to disjunctive queries: If $\Phi_1 \vee \ldots \vee \Phi_n$ is known to be true in an instance r and the formulas $\Phi_1, \ldots, \Phi_{n-1}$ are known to be false in r, then Φ_n must be true in r. Consequently, enhancements of $\mathscr{L}_\mathrm{q}$ must prevent disjunctive structures in queries if static inference control is desired. We propose a query language $\mathscr{L}_\mathrm{q}^{cn}$ by adding conjunction and negation such that disjunction cannot be simulated. This is achieved by restricting negation to existential-*R*-sentences.

Definition 2 (Query language with conjunction and negation). *The query language* $\mathscr{L}_\mathrm{q}^{cn}$ $(\subseteq \mathscr{L}_\mathrm{q}^{max})$ *is inductively defined as follows: (1) If* $\Phi \in \mathscr{L}_\mathrm{q}$ *then* $\Phi \in \mathscr{L}_\mathrm{q}^{cn}$*; (2) if* $\Phi \in \mathscr{L}_\mathrm{q}$ *then* $\neg\Phi \in \mathscr{L}_\mathrm{q}^{cn}$*; (3) if* $\Phi_1, \Phi_2 \in \mathscr{L}_\mathrm{q}^{cn}$ *then* $\Phi_1 \wedge \Phi_2 \in \mathscr{L}_\mathrm{q}^{cn}$*.*

An answer to a query from $\mathscr{L}_\mathrm{q}^{cn}$ gives the user an "all or nothing" information: If each conjunct is true in the database instance, then the whole query is true; otherwise, the whole query is false. To provide a more differentiated answer in case the query is false, we suggest to consider a query $\Phi \equiv \Phi_1 \wedge \ldots \wedge \Phi_n$ from $\mathscr{L}_\mathrm{q}^{cn}$ as a sequence $\langle \Phi_1, \ldots, \Phi_n \rangle$ of queries from $\mathscr{L}_\mathrm{q}$. Thus, the answer to Φ is a sequence $\langle ans_1, \ldots, ans_n \rangle$. The resulting censor for queries from $\mathscr{L}_\mathrm{q}^{cn}$ is denoted by $censor_{stat}^{cn}$.

Theorem 1. *The CQE induced by* $censor_{stat}^{cn}$*, hereafter called* cqe_{stat}^{cn}*, preserves confidentiality in the sense of Def. 1.*

3.2 The Policy Language

3.2.1 Revising the Definition of Fact Schemas

According to Sect. 2, the security administrator must restrict to facts when declaring a policy. Recall $fs(bank\_db)$ from Example 1: For protecting the association between an account number and the account holder, also the corresponding bank has to be protected.

In the following, we present an alternative definition of fact schemas leading to a greater flexibility in declaring policies while still guaranteeing confidentiality when these policies are enforced. This definition is driven by two ideas: It suffices to include a *subset* of the key into a fact schema; each *single* attribute suits as fact schema—whether or not it is a key attribute.

Definition 3 (Alternative fact schemas). *Let* $\langle R, \mathscr{U}, \Sigma \rangle$ *be a relation schema in ONF. The left-hand side of an fd* $\sigma \in \Sigma$ *is denoted by* $lhs(\sigma)$*. The* alternative set of fact schemas of *RS is then defined by*

$$fs_{alt}(RS) = \{A \mid A \in \mathscr{U}\} \cup \{\mathscr{A} \mid exists\ \sigma \in \Sigma : \mathscr{A} \subseteq lhs(\sigma)\} \cup \{\mathscr{A}B \mid exists\ \sigma \in \Sigma\ such\ that\ \mathscr{A} \subseteq lhs(\sigma)\ and\ B \in \mathscr{U} \setminus lhs(\sigma)\}.$$

Theorem 2. *When exchanging the fact schema definition* $fs(RS)$ *by* $fs_{alt}(RS)$ *from Def. 3,* cqe_{stat} *still preserves confidentiality in the sense of Def. 1.*

Reconsidering Example 1, we get the following set of alternative fact schemas:

$$fs_{alt}(bank\_db) = fs(bank\_db) \cup \{\{bank\}, \{acc\_no\}, \{acc\_holder\}, \{balance\}, \{bank, acc\_holder\}, \{bank, balance\}, \{acc\_no, acc\_holder\}, \{acc\_no, balance\}\}.$$

It is now possible to protect the association between an account number and the account holder without protecting the bank.

In general, for a relation schema in ONF with n attributes and a key of size k, the original facts definition yields $n-k+1$ different fact schemas, whereas the alternative definition yields $2^k(n-k+1)-1$ different facts schemas.

3.2.2 Introducing Disjunction

Like the query language $\mathscr{L}_{\mathrm{q}}$, also the policy language $\mathscr{L}_{\mathrm{ps}}(\subseteq \mathscr{L}_{\mathrm{ps}}^{max})$ in [8] is restricted to existential-R-sentences. Adding negation or conjunction to $\mathscr{L}_{\mathrm{ps}}$ possibly enables the user to disclose secrets as illustrated by the following examples (which are based on Example 1). We thus propose a policy language by adding disjunction.

Example 3. Regarding negation, we consider the following policy and query:

$$pot\_sec = \{\neg(\exists X_{bal})bank\_db(\text{Bank of Springfield}, 213456, \text{Jones}, X_{bal})\}$$
$$\Phi \equiv (\exists X_{bal})bank\_db(\text{Bank of Springfield}, 213456, \text{Green}, X_{bal})$$

Obviously, Φ is answered correctly by cqe_{stat}. However, by employing the a priori knowledge Σ, the user knows that each instantiation of $(bank, acc\_no)$ is unique. The correct answer to Φ thereby implies the potential secret.

Example 4. Regarding conjunction, we consider the following policy and queries:

$$pot\_sec = \{\ (\exists X_{ah})(\exists X_{bal})bank\_db(\text{Bank of Springfield}, 123654, X_{ah}, X_{bal}) \wedge (\exists X_{ah})(\exists X_{bal})bank\_db(\text{Gotham City Bank}, 312564, X_{ah}, X_{bal})\}$$
$$\Phi_1 \equiv (\exists X_{ah})(\exists X_{bal})bank\_db(\text{Bank of Springfield}, 123654, X_{ah}, X_{bal})$$
$$\Phi_2 \equiv (\exists X_{ah})(\exists X_{bal})bank\_db(\text{Gotham City Bank}, 312564, X_{ah}, X_{bal})$$

Obviously, cqe_{stat} answers both Φ_1 and Φ_2 correctly. However, combining both answers implies the potential secret.

Definition 4 (Disjunctive policy language). *The policy language $\mathscr{L}_{\mathrm{ps}}^{d}$ $(\subseteq \mathscr{L}_{\mathrm{ps}}^{max})$ is inductively defined as follows: (1) If $\Psi \in \mathscr{L}_{\mathrm{ps}}$ then $\Psi \in \mathscr{L}_{\mathrm{ps}}^{d}$; (2) if $\Psi_1, \Psi_2 \in \mathscr{L}_{\mathrm{ps}}^{d}$ then $\Psi_1 \vee \Psi_2 \in \mathscr{L}_{\mathrm{ps}}^{d}$.*

Theorem 3. *The CQE emerging from cqe_{stat} by substituting $\mathscr{L}_{\mathrm{ps}}$ with $\mathscr{L}_{\mathrm{ps}}^{d}$, hereafter denoted with cqe_{stat}^{d}, preserves confidentiality in the sense of Def. 1.*

3.2.3 Introducing Free Variables

So far, elements of the policy language refer to tuples, subtuples, or disjunctions of (sub-)tuples. For practical purposes, this restriction might be unsatisfactory. Reconsider the schema from Example 1 and suppose a large instance of *bank_db*. If the Bank of Springfield wants to keep the connections between account numbers and account holders confidential, the security administrator has to add formulas of the form $(\exists X_{bal})bank\_db(\text{Bank of Springfield}, N, H, X_{bal})$ to the policy for every single constant combination of account number N and account holder H actually occurring in *bank_db*.

This is tedious and compromises the confidentiality: If the user knows that each *actually occurring* instantiation of a set of attributes is protected, he can simply determine these secrets from the policy (which is supposed to be public).

Protecting *every* constant combination of N and H (whether occurring in *bank_db* or not) requires to introduce free variables, since the underlying universe is supposed to be infinite. More specifically, we denote the policy language emerging from $\mathscr{L}_{\text{ps}}$ by introducing free variables with $\mathscr{L}_{\text{ps}}^{f}$. An element from $\mathscr{L}_{\text{ps}}^{f}$ is denoted by $\Psi(\vec{V})$ with $\vec{V} = (X_1, \ldots, X_l)$ being the vector of free variables occurring in $\Psi(\vec{V})$. A potential secret with free variables $\Psi(\vec{V}) \in \mathscr{L}_{\text{ps}}^{f}$ is expanded to the (infinite) set $ex(\Psi(\vec{V})) \subset \mathscr{L}_{\text{ps}}$ by substituting the free variables $\vec{V}$ with every possible constant combination. An element from $ex(\Psi(\vec{V}))$ is denoted by $\Psi(\vec{c})$ with $\vec{c}$ being a vector of constants. The expansion of a policy $pot\_sec \subset \mathscr{L}_{\text{ps}}^{f}$ is defined by $ex(pot\_sec) = \bigcup_{\Psi(\vec{V}) \in pot\_sec} ex(\Psi(\vec{V})) \subset \mathscr{L}_{\text{ps}}$. We now adapt the definition of the static censor (2) and the definition of confidentiality preservation to $\mathscr{L}_{\text{ps}}^{f}$:

$$censor_{stat}^{f}(pot\_sec, \Phi) := (\text{exists } \Psi(\vec{c}))(\Psi(\vec{c}) \in ex(pot\_sec) \text{ and } \Phi \models \Psi(\vec{c}))$$

Definition 5 (Confidentiality preservation for $\mathscr{L}_{\text{ps}}^{f}$). *This definition emerges from Def. 1 by replacing each Ψ with $\Psi(\vec{c})$ and $\Psi \in pot\_sec$ with $\Psi(\vec{c}) \in ex(pot\_sec)$.*

Theorem 4. *The CQE induced by $censor_{stat}^{f}$, hereafter called cqe_{stat}^{f}, preserves confidentiality in the sense of Def. 5.*

Unfortunately, $censor_{stat}^{f}$ has no straightforward algorithmic interpretation, since it has to check the elements of an infinite policy. We therefore propose an alternative censor, $censor_{stat}^{f,alt}$, which is defined in an algorithmic way and prove it equivalent to $censor_{stat}^{f}$. In the following, $\chi[A_i]$ denotes the instantiation of attribute A_i in the existential-R-sentence χ, e. g., if $\Psi(X_f) \equiv (\exists X_b) R(a, X_b, X_f)$, then $\Psi(X_f)[A_1] = a$, $\Psi(X_f)[A_2] = X_b$, and $\Psi(X_f)[A_3] = X_f$.

$$\begin{aligned} censor_{stat}^{f,alt}(pot\_sec, \Phi) := {} & (\text{exists } \Psi(\vec{V}))(\Psi(\vec{V}) \in pot\_sec \text{ and for all } A \in \mathscr{U} : \\ & \text{if } \Psi(\vec{V})[A] \in Const, \text{ then } \Phi[A] = \Psi(\vec{V})[A] \text{ and} \qquad (3) \\ & \text{if } \Psi(\vec{V})[A] \text{ is a free variable, then } \Phi[A] \in Const) \qquad (4) \end{aligned}$$

Lemma 1. *Let $\Phi \in \mathscr{L}_{\text{q}}$ be a query and $\Psi(\vec{V}) \in \mathscr{L}_{\text{ps}}^{f}$ a potential secret with free variables. Then, there exists a vector of constants $\vec{c}$ with $\Psi(\vec{c}) \in ex(\Psi(\vec{V}))$ such that $\Phi \models \Psi(\vec{c})$ if and only if for all attributes $A \in \mathscr{U}$ (3) and (4) hold.*

Theorem 5. *The CQE induced by $censor_{stat}^{f,alt}$, hereafter called $cqe_{stat}^{f,alt}$, preserves confidentiality in the sense of Def. 5.*

Finally, we justify that $\mathscr{L}_{\text{ps}}^{d}$ and $\mathscr{L}_{\text{ps}}^{f}$ are "compatible". Consider the policy language $\mathscr{L}_{\text{ps}}^{df}$ which is enhanced with disjunction *and* free variables. Following the proof of Theorem 3, under the given assumptions, static CQE is equivalent on $pot\_sec = \{\Psi_1, \ldots, \Psi_l\}$ and $pot\_sec' = \{\Psi_1 \vee \ldots \vee \Psi_l\}$ with $\Psi_i \in \mathscr{L}_{\text{ps}}$. The same argumentation can be applied if $\Psi_i \in \mathscr{L}_{\text{ps}}^{f}$. Thus, for each policy $pot\_sec \subset \mathscr{L}_{\text{ps}}^{df}$ we can break up each disjunctive secret into atomic secrets.

3.3 Limits of the Optimization

In Subsect. 3.1, we pointed out that using disjunction in queries can be harmful regarding confidentiality preservation. In particular, disjunctive structures can be interpreted as implicative structures, e. g., $\chi_1 \vee \chi_2 \equiv \neg\chi_1 \implies \chi_2$. If a purely static CQE is desired, disjunctive structures must be avoided in queries at all.

Regarding facts, Def. 3 requires each policy element to protect either a single attribute value or *at least* one key attribute value together with *at most* one non-key attribute value. The combination of two or more non-key attribute values could be disclosed with separate queries, each of which asking for one of the non-key attribute values in combination with the key value. E. g., consider a key $\mathscr{K}$ and two non-key attributes N_1 and N_2; if an element of the policy protects a value combination of $\mathscr{K}N_1N_2$, then the user can first ask for the value combination $\mathscr{K}N_1$ and later for the value combination $\mathscr{K}N_2$. Considered separately, neither of the queries discloses a potential secret; however, exploiting the uniqueness property of the key leads to the disclosure of the value combination of $\mathscr{K}N_1N_2$.

As demonstrated by Examples 3 and 4 in Subsect. 3.2, also the policy language cannot be enhanced arbitrarily. Negative potential secrets possibly enable the user to employ fds and conjunctive secrets can be disclosed "piece by piece". Thus, only disjunction can be added to the policy language without problems.

4 Implementing Static Inference Control in SQL

Implementations of static censors do not need external theorem provers but can utilize the functionality of the database management system. We assume that the potential secrets are encoded as tuples of a classification instance *R_ps* by replacing existentially quantified variables with the "new" symbol #. E. g., $\Psi \equiv (\exists X)R(a,X)$ is represented in *R_ps* by $R(a,\#)$. Let $\Phi \in \mathscr{L}_\mathrm{q}$ be a query, $A\_1,\ldots,A\_l$ the attributes being instantiated by constants $a\_1,\ldots,a\_l$ in Φ, $B\_1,\ldots,B\_m$ the attributes being instantiated by existentially quantified variables in Φ, and $pot\_sec \subset \mathscr{L}_\mathrm{ps}$ a policy. Elementary considerations indicate that $\Phi \models \Psi$ (as needed for $censor_{stat}$) holds for some $\Psi \in pot\_sec$ if and only if the following SQL statement yields a number greater than zero (adaptions for $censor^{cn}_{stat}$ and $censor^{d}_{stat}$ are straightforward):

```
SELECT COUNT(*) FROM R_ps
WHERE (A_1 = 'a_1' OR A_1 ='#') AND (A_2 = 'a_2' OR A_2 ='#')
  AND ... AND (A_l = 'a_l' OR A_l = '#')
  AND (B_1 = '#') AND (B_2 = '#') AND ... AND (B_m = '#')
```

Considering $censor^{f,alt}_{stat}$, we encode free variables in *R_ps* by a "new" symbol $\sim$. E. g., $\Psi \equiv (\exists X_b)R(a,X_b,X_f)$ is represented in *R_ps* by $R(a,\#,\sim)$. Let Φ be defined as above, and $pot\_sec \subset \mathscr{L}^f_\mathrm{ps}$ a policy. $\Phi \models \Psi'$ holds for some $\Psi' \in ex(pot\_sec)$ if and only if the following SQL statement yields a number greater than zero:

```
SELECT COUNT(*) FROM R_ps
WHERE (A_1 = 'a_1' OR A_1 = '#' OR A_1 = '~')
  AND ... AND (A_l = 'a_l' OR A_l = '#' OR A_1 = '~')
  AND (B_1 = '#') AND (B_2 = '#') AND ... AND (B_m = '#')
```

5 Towards an Optimized Inference Control System

We can put together the results of Sect. 3, i. e., substituting $\mathscr{L}_{\mathrm{q}}$ with $\mathscr{L}_{\mathrm{q}}^{cn}$ and $\mathscr{L}_{\mathrm{ps}}$ with $\mathscr{L}_{\mathrm{ps}}^{df}$, exchanging $fs(RS)$ by $fs_{alt}(RS)$, and sequencing conjunctions; the resulting CQE, denoted by cqe_{stat}^{opt}, preserves confidentiality in the sense of Def. 5.

Especially for a database user, the query language $\mathscr{L}_{\mathrm{q}}^{cn}$ is still unsatisfactory. To improve the situation, we introduce an algorithm that principally accepts each query Φ from $\mathscr{L}_{\mathrm{q}}^{max}$ but, if necessary, transforms Φ into a "stronger" query $\Phi_{cn} \in \mathscr{L}_{\mathrm{q}}^{cn}$ with $\Phi_{cn} \models \Phi$ (but possibly $\Phi \not\models \Phi_{cn}$). Using this algorithm, we sketch an interactive system, providing expressive policy and query languages on the one hand, and (if possible) offering static inference control on the other hand. The idea to transform a "harmful" query into a "harmless" query is related to the concept of query modification, introduced by Stonebraker/Wong [22]. While query modification suitably appends a conjunct to each query, our approach rearranges the given syntactic structure of the query. Our algorithm expects a query $\Phi \in \mathscr{L}_{\mathrm{q}}^{max}$, an a priori user knowledge log_0, an instance r, and a policy *pot_sec* as input. It works as follows:

(1) Convert Φ into prenex disjunctive normal form $\Phi_{PDNF} \equiv (\exists X_1)\ldots(\exists X_l)\ (\bigvee_{i=1}^{m}(\bigwedge_{j=1}^{n_i}\varphi_j))$, where φ_j denotes a (possibly negated) atomic formula.
(2) Rearrange Φ_{PDNF} into $\Phi_{DNF} \equiv \bigvee_{i=1}^{m}(\bigwedge_{j=1}^{n_i}(\exists X_{j_1})\ldots(\exists X_{j_l})\varphi_j)$, where X_{j_k} occurs in φ_j. This step is correct because of the assumption that each existentially quantified variable occurs only once in the formula.
(3) Transform Φ_{DNF} into $\Phi_{cn} :\equiv \bigwedge_{i=1}^{m}(\bigwedge_{j=1}^{n_i}(\exists X_{j_1})\ldots(\exists X_{j_l})\varphi_j)$. Note that $\Phi_{cn} \not\equiv \Phi$. However, it can easily be verified that $\Phi_{cn} \models \Phi_{DNF}$ and thus $\Phi_{cn} \models \Phi$.
(4) Return $cqe_{stat}^{cn}(\langle\Phi_{cn}\rangle, log_0)(r, pot\_sec)$.

An interactive inference control system now roughly proceeds in two phases. The database instance r is assumed to be set up in advance. Initially, the system starts in "static inference control mode" (SIC mode), which means that the static censors are used when answering user queries (analogously, in "dynamic inference control mode" (DIC mode), only the non-static censors are used).

Policy declaration phase: The security administrator declares $pot\_sec = \{\Psi_1, \ldots, \Psi_m\}$ with $\Psi_i \in \mathscr{L}_{\mathrm{ps}}^{max}$. If *pot_sec* contains a $\Psi_i \notin \mathscr{L}_{\mathrm{ps}}^{df}$, a static inference control cannot be performed later on. For every such Ψ_i, the security administrator can choose between the following actions: a) withdraw Ψ_i; b) affirm Ψ_i; in this case, the system completely switches to DIC mode.

Query phase (usually performed repeatedly): A user sends a query $\Phi \in \mathscr{L}_{\mathrm{q}}^{max}$ to the database. If the system is in DIC mode, Φ is answered. If the system is in SIC mode and $\Phi \notin \mathscr{L}_{\mathrm{q}}^{cn}$, the user can choose between the following actions: a) withdraw Φ; b) affirm Φ; in this case, the system completely switches to DIC mode and Φ is answered; c) accept the rewrite suggestion Φ_{cn} (according to the above sketched algorithm); in this case, Φ_{cn} is answered instead of Φ.

6 Conclusion and Future Work

We investigated efficient inference control enforcing policies for closed queries in relational databases by identifying situations in which it is possible to apply static CQE and by presenting suitable SQL implementations. We proposed an interactive inference control system, issuing database users and security administrators with flexible languages for expressing queries and potential secrets. These languages have been enhanced compared to the static CQE in [8] while it is still possible to employ static censors guaranteeing feasible runtime.

However, our approach is not meant to be an exhaustive optimization of CQE in relational databases, but should be seen as a step in this direction. Further development could deal with global semantic constraints (such as inclusion dependencies), other types of local semantic constraints (such as multivalued dependencies), free variables in the query language to express open queries, and alternative CQE enforcement methods (lying and combined method; see [4]).

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Bertino, E., Sandhu, R.: Database security – concepts, approaches, and challenges. IEEE Trans. Dependable Sec. Comput. 2(1), 2–18 (2005)
3. Biskup, J.: Boyce-Codd normal form and object normal forms. Inf. Process. Lett. 32(1), 29–33 (1989)
4. Biskup, J., Bonatti, P.: Controlled query evaluation for enforcing confidentiality in complete information systems. Int. J. Inf. Sec. 3(1), 14–27 (2004)
5. Biskup, J., Bonatti, P.: Controlled query evaluation with open queries for a decidable relational submodel. Ann. Math. Artif. Intell. 50, 39–77 (2007)
6. Biskup, J., Bonatti, P.A.: Lying versus refusal for known potential secrets. Data Knowl. Eng. 38, 199–222 (2001)
7. Biskup, J., Bonatti, P.A.: Controlled query evaluation for known policies by combining lying and refusal. Ann. Math. Artif. Intell. 40, 37–62 (2004)
8. Biskup, J., Embley, D.W., Lochner, J.-H.: Reducing inference control to access control for normalized database schemas. Inf. Process. Lett. 106(1), 8–12 (2008)
9. Biskup, J., Lochner, J.-H.: Enforcing confidentiality in relational databases by reducing inference control to access control. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 407–422. Springer, Heidelberg (2007)
10. Bonatti, P., Kraus, S., Subrahmanian, V.S.: Foundations of secure deductive databases. IEEE Trans. Knowl. Data Eng. 7(3), 406–422 (1995)
11. Brodsky, A., Farkas, C., Jajodia, S.: Secure databases: constraints, inference channels, and monitoring disclosures. IEEE Trans. Knowl. Data Eng. 12(6), 900–919 (2000)
12. Byun, J.W., Bertino, E.: Micro-views, or on how to protect privacy while enhancing data usability—concepts and challenges. ACM SIGMOD Record 35(1), 9–13 (2006)
13. Cuppens, F., Gabillon, A.: Cover story management. Data Knowl. Eng. 37(2), 177–201 (2001)
14. Dawson, S., De Capitani di Vimercati, S., Samarati, P.: Specification and enforcement of classification and inference constraints. In: IEEE Symposium on Security and Privacy, pp. 181–195. IEEE Computer Society, Los Alamitos (1999)

15. Farkas, C., Jajodia, S.: The inference problem: a survey. SIGKDD Explorations 4(2), 6–11 (2002)
16. Griffiths, P.P., Wade, B.W.: An authorization mechanism for a relational database system. ACM Trans. Database Syst. 1(3), 242–255 (1976)
17. Jajodia, S., Sandhu, R.S.: Toward a multilevel secure relational data model. In: SIGMOD Conference, pp. 50–59. ACM, New York (1991)
18. Lunt, T.F., Denning, D.E., Schell, R.R., Heckman, M., Shockley, W.R.: The seaview security model. IEEE Trans. Software Eng. 16(6), 593–607 (1990)
19. Rjaibi, W., Bird, P.: A multi-purpose implementation of mandatory access control in relational database management systems. In: VLDB, pp. 1010–1020. ACM, New York (2004)
20. Sandhu, R.: Lattice-based access control models. Computer 26(11), 9–19 (1993)
21. Sicherman, G.L., de Jonge, W., van de Riet, R.P.: Answering queries without revealing secrets. ACM Trans. Database Syst. 8(1), 41–59 (1983)
22. Stonebraker, M., Wong, E.: Access control in a relational data base management system by query modification. In: ACM/CSC-ER, pp. 180–186. ACM, New York (1974)